```
/**
 * SWFUpload: http://www.swfupload.org,
http://swfupload.googlecode.com
 *
 * mmSWFUpload 1.0: Flash upload dialog -
http://profandesign.se/swfupload/,
http://www.vinterwebb.se/
 *
 * SWFUpload is (c) 2006-2007 Lars Huring, Olov Nilzén and
Mammon Media and is released under the MIT License:
 * http://www.opensource.org/licenses/mit-license.php
 *
 * SWFUpload 2 is (c) 2007-2008 Jake Roberts and is
released under the MIT License:
 * http://www.opensource.org/licenses/mit-license.php
 *
 */


/* ******************* */
/* Constructor & Init  */
/* ******************* */
var SWFUpload;

if (SWFUpload == undefined) {
	SWFUpload = function (settings) {
		this.initSWFUpload(settings);
	};
}

SWFUpload.prototype.initSWFUpload = function (settings) {
	try {
		this.customSettings = {}; // A container where
developers can place their own settings associated with
this instance.
		this.settings = settings;
		this.eventQueue = [];
		this.movieName = "SWFUpload_" +
SWFUpload.movieCount++;
		this.movieElement = null;


		// Setup global control tracking
		SWFUpload.instances[this.movieName] = this;

		// Load the settings.  Load the Flash movie.
		this.initSettings();
```

```
			this.loadFlash();
			this.displayDebugInfo();
		} catch (ex) {
			delete SWFUpload.instances[this.movieName];
			throw ex;
		}
	};

	/* *************** */
	/* Static Members  */
	/* *************** */
	SWFUpload.instances = {};
	SWFUpload.movieCount = 0;
	SWFUpload.version = "2.2.0 Beta 5 2008-01-29";
	SWFUpload.QUEUE_ERROR = {
		QUEUE_LIMIT_EXCEEDED			: -100,
		FILE_EXCEEDS_SIZE_LIMIT		: -110,
		ZERO_BYTE_FILE				: -120,
		INVALID_FILETYPE			: -130
	};
	SWFUpload.UPLOAD_ERROR = {
		HTTP_ERROR				: -200,
		MISSING_UPLOAD_URL		: -210,
		IO_ERROR				: -220,
		SECURITY_ERROR				: -230,
		UPLOAD_LIMIT_EXCEEDED			: -240,
		UPLOAD_FAILED				: -250,
		SPECIFIED_FILE_ID_NOT_FOUND		: -260,
		FILE_VALIDATION_FAILED			: -270,
		FILE_CANCELLED				: -280,
		UPLOAD_STOPPED				: -290
	};
	SWFUpload.FILE_STATUS = {
		QUEUED		: -1,
		IN_PROGRESS	: -2,
		ERROR		: -3,
		COMPLETE	: -4,
		CANCELLED	: -5
	};
	SWFUpload.BUTTON_ACTION = {
		SELECT_FILE  : -100,
		SELECT_FILES : -110,
		START_UPLOAD : -120
	};
	SWFUpload.CURSOR = {
		ARROW : -1,
		HAND : -2
```

```
};
SWFUpload.WINDOW_MODE = {
	WINDOW : "window",
	TRANSPARENT : "transparent",
	OPAQUE : "opaque"
};

/* ******************** */
/* Instance Members  */
/* ******************** */

// Private: initSettings ensures that all the
// settings are set, getting a default value if one was not
assigned.
SWFUpload.prototype.initSettings = function () {
	this.ensureDefault = function (settingName,
defaultValue) {
		this.settings[settingName] =
(this.settings[settingName] == undefined) ? defaultValue :
this.settings[settingName];
	};

	// Upload backend settings
	this.ensureDefault("upload_url", "");
	this.ensureDefault("file_post_name", "Filedata");
	this.ensureDefault("post_params", {});
	this.ensureDefault("use_query_string", false);
	this.ensureDefault("requeue_on_error", false);
	this.ensureDefault("http_success", []);

	// File Settings
	this.ensureDefault("file_types", "*.*");
	this.ensureDefault("file_types_description", "All
Files");
	this.ensureDefault("file_size_limit", 0);	//
Default zero means "unlimited"
	this.ensureDefault("file_upload_limit", 0);
	this.ensureDefault("file_queue_limit", 0);

	// Flash Settings
	this.ensureDefault("flash_url", "swfupload.swf");
	this.ensureDefault("prevent_swf_caching", true);

	// Button Settings
	this.ensureDefault("button_image_url", "");
	this.ensureDefault("button_width", 1);
	this.ensureDefault("button_height", 1);
	this.ensureDefault("button_text", "");
	this.ensureDefault("button_text_style", "color: 
#000000; font-size: 16pt;");
	this.ensureDefault("button_text_top_padding", 0);
	this.ensureDefault("button_text_left_padding", 0);
	this.ensureDefault("button_action", 
SWFUpload.BUTTON_ACTION.SELECT_FILES);
	this.ensureDefault("button_disabled", false);
	this.ensureDefault("button_placeholder_id", "");
	this.ensureDefault("button_cursor", 
SWFUpload.CURSOR.ARROW);
	this.ensureDefault("button_window_mode", 
SWFUpload.WINDOW_MODE.WINDOW);

	// Debug Settings
	this.ensureDefault("debug", false);
	this.settings.debug_enabled = this.settings.debug; // 
Here to maintain v2 API

	// Event Handlers
	this.settings.return_upload_start_handler = 
this.returnUploadStart;
	this.ensureDefault("swfupload_loaded_handler", null);
	this.ensureDefault("file_dialog_start_handler", 
null);
	this.ensureDefault("file_queued_handler", null);
	this.ensureDefault("file_queue_error_handler", null);
	this.ensureDefault("file_dialog_complete_handler", 
null);

	this.ensureDefault("upload_start_handler", null);
	this.ensureDefault("upload_progress_handler", null);
	this.ensureDefault("upload_error_handler", null);
	this.ensureDefault("upload_success_handler", null);
	this.ensureDefault("upload_complete_handler", null);

	this.ensureDefault("debug_handler", 
this.debugMessage);

	this.ensureDefault("custom_settings", {});

	// Other settings
	this.customSettings = this.settings.custom_settings;

	// Update the flash url if needed
	if (!!this.settings.prevent_swf_caching) {
		this.settings.flash_url =
this.settings.flash_url +
(this.settings.flash_url.indexOf("?") < 0 ? "?" : "&") +
"preventswfcaching=" + new Date().getTime();
	}

	delete this.ensureDefault;
};

// Private: loadFlash replaces the button_placeholder
element with the flash movie.
SWFUpload.prototype.loadFlash = function () {
	var targetElement, tempParent;

	// Make sure an element with the ID we are going to
use doesn't already exist
	if (document.getElementById(this.movieName) !== null)
{
		throw "ID " + this.movieName + " is already in
use. The Flash Object could not be added";
	}

	// Get the element where we will be placing the flash
movie
	targetElement =
document.getElementById(this.settings.button_placeholder_id
);

	if (targetElement == undefined) {
		throw "Could not find the placeholder element:
" + this.settings.button_placeholder_id;
	}

	// Append the container and load the flash
	tempParent = document.createElement("div");
	tempParent.innerHTML = this.getFlashHTML();	// Using
innerHTML is non-standard but the only sensible way to
dynamically add Flash in IE (and maybe other browsers)
	targetElement.parentNode.replaceChild(tempParent.firs
tChild, targetElement);

	// Fix IE Flash/Form bug
	if (window[this.movieName] == undefined) {
		window[this.movieName] =
this.getMovieElement();
	}
```

```
};

// Private: getFlashHTML generates the object tag needed to
embed the flash in to the document
SWFUpload.prototype.getFlashHTML = function () {
	// Flash Satay object syntax:
http://www.alistapart.com/articles/flashsatay
	return ['<object id="', this.movieName, '"
type="application/x-shockwave-flash" data="',
this.settings.flash_url, '" width="',
this.settings.button_width, '" height="',
this.settings.button_height, '" class="swfupload">',
				'<param name="wmode" value="',
this.settings.button_window_mode, '" />',
				'<param name="movie" value="',
this.settings.flash_url, '" />',
				'<param name="quality"
value="high" />',
				'<param name="menu" value="false"
/>',
				'<param name="allowScriptAccess"
value="always" />',
				'<param name="flashvars" value="'
+ this.getFlashVars() + '" />',
				'</object>'].join("");
};

// Private: getFlashVars builds the parameter string that
will be passed
// to flash in the flashvars param.
SWFUpload.prototype.getFlashVars = function () {
	// Build a string from the post param object
	var paramString = this.buildParamString();
	var httpSuccessString =
this.settings.http_success.join(",");

	// Build the parameter string
	return ["movieName=",
encodeURIComponent(this.movieName),
			"&amp;uploadURL=",
encodeURIComponent(this.settings.upload_url),
			"&amp;useQueryString=",
encodeURIComponent(this.settings.use_query_string),
			"&amp;requeueOnError=",
encodeURIComponent(this.settings.requeue_on_error),
			"&amp;httpSuccess=",
encodeURIComponent(httpSuccessString),
                   "&amp;params=",
encodeURIComponent(paramString),
                   "&amp;filePostName=",
encodeURIComponent(this.settings.file_post_name),
                   "&amp;fileTypes=",
encodeURIComponent(this.settings.file_types),
                   "&amp;fileTypesDescription=",
encodeURIComponent(this.settings.file_types_description),
                   "&amp;fileSizeLimit=",
encodeURIComponent(this.settings.file_size_limit),
                   "&amp;fileUploadLimit=",
encodeURIComponent(this.settings.file_upload_limit),
                   "&amp;fileQueueLimit=",
encodeURIComponent(this.settings.file_queue_limit),
                   "&amp;debugEnabled=",
encodeURIComponent(this.settings.debug_enabled),
                   "&amp;buttonImageURL=",
encodeURIComponent(this.settings.button_image_url),
                   "&amp;buttonWidth=",
encodeURIComponent(this.settings.button_width),
                   "&amp;buttonHeight=",
encodeURIComponent(this.settings.button_height),
                   "&amp;buttonText=",
encodeURIComponent(this.settings.button_text),
                   "&amp;buttonTextTopPadding=",
encodeURIComponent(this.settings.button_text_top_padding),
                   "&amp;buttonTextLeftPadding=",
encodeURIComponent(this.settings.button_text_left_padding),
                   "&amp;buttonTextStyle=",
encodeURIComponent(this.settings.button_text_style),
                   "&amp;buttonAction=",
encodeURIComponent(this.settings.button_action),
                   "&amp;buttonDisabled=",
encodeURIComponent(this.settings.button_disabled),
                   "&amp;buttonCursor=",
encodeURIComponent(this.settings.button_cursor)
             ].join("");
};

// Public: getMovieElement retrieves the DOM reference to
the Flash element added by SWFUpload
// The element is cached after the first lookup
SWFUpload.prototype.getMovieElement = function () {
      if (this.movieElement == undefined) {
             this.movieElement =
document.getElementById(this.movieName);
      }
```

```
	if (this.movieElement === null) {
		throw "Could not find Flash element";
	}

	return this.movieElement;
};

// Private: buildParamString takes the name/value pairs in
the post_params setting object
// and joins them up in to a string formatted
"name=value&amp;name=value"
SWFUpload.prototype.buildParamString = function () {
	var postParams = this.settings.post_params;
	var paramStringPairs = [];

	if (typeof(postParams) === "object") {
		for (var name in postParams) {
			if (postParams.hasOwnProperty(name)) {

	paramStringPairs.push(encodeURIComponent(name.toStrin
g()) + "=" +
encodeURIComponent(postParams[name].toString()));
			}
		}
	}

	return paramStringPairs.join("&amp;");
};

// Public: Used to remove a SWFUpload instance from the
page. This method strives to remove
// all references to the SWF, and other objects so memory
is properly freed.
// Returns true if everything was destroyed. Returns a
false if a failure occurs leaving SWFUpload in an
inconsistant state.
// Credits: Major improvements provided by steffen
SWFUpload.prototype.destroy = function () {
	try {
		// Make sure Flash is done before we try to
remove it
		this.cancelUpload(null, false);


		// Remove the SWFUpload DOM nodes
		var movieElement = null;
```

```
			movieElement = this.getMovieElement();

			if (movieElement &&
typeof(movieElement.CallFunction) === "unknown") { // We
only want to do this in IE
				// Loop through all the movie's
properties and remove all function references (DOM/JS IE
6/7 memory leak workaround)
				for (var i in movieElement) {
					try {
						if (typeof(movieElement[i])
=== "function") {
							movieElement[i] =
null;
						}
					} catch (ex1) {}
				}

				// Remove the Movie Element from the
page
				try {

	movieElement.parentNode.removeChild(movieElement);
				} catch (ex) {}
			}

			// Remove IE form fix reference
			window[this.movieName] = null;

			// Destroy other references
			SWFUpload.instances[this.movieName] = null;
			delete SWFUpload.instances[this.movieName];

			this.movieElement = null;
			this.settings = null;
			this.customSettings = null;
			this.eventQueue = null;
			this.movieName = null;


			return true;
		} catch (ex2) {
			return false;
		}
};
```

```
// Public: displayDebugInfo prints out settings and
configuration
// information about this SWFUpload instance.
// This function (and any references to it) can be deleted
when placing
// SWFUpload in production.
SWFUpload.prototype.displayDebugInfo = function () {
	this.debug(
		[
			"---SWFUpload Instance Info---\n",
			"Version: ", SWFUpload.version, "\n",
			"Movie Name: ", this.movieName, "\n",
			"Settings:\n",
			"\t", "upload_url:               ",
this.settings.upload_url, "\n",
			"\t", "flash_url:                ",
this.settings.flash_url, "\n",
			"\t", "use_query_string:         ",
this.settings.use_query_string.toString(), "\n",
			"\t", "requeue_on_error:         ",
this.settings.requeue_on_error.toString(), "\n",
			"\t", "http_success:             ",
this.settings.http_success.join(", "), "\n",
			"\t", "file_post_name:           ",
this.settings.file_post_name, "\n",
			"\t", "post_params:              ",
this.settings.post_params.toString(), "\n",
			"\t", "file_types:               ",
this.settings.file_types, "\n",
			"\t", "file_types_description:   ",
this.settings.file_types_description, "\n",
			"\t", "file_size_limit:          ",
this.settings.file_size_limit, "\n",
			"\t", "file_upload_limit:        ",
this.settings.file_upload_limit, "\n",
			"\t", "file_queue_limit:         ",
this.settings.file_queue_limit, "\n",
			"\t", "debug:                    ",
this.settings.debug.toString(), "\n",

			"\t", "prevent_swf_caching:      ",
this.settings.prevent_swf_caching.toString(), "\n",

			"\t", "button_placeholder_id:    ",
this.settings.button_placeholder_id.toString(), "\n",
			"\t", "button_image_url:         ",
this.settings.button_image_url.toString(), "\n",
                   "\t", "button_width:              ",
this.settings.button_width.toString(), "\n",
                   "\t", "button_height:             ",
this.settings.button_height.toString(), "\n",
                   "\t", "button_text:               ",
this.settings.button_text.toString(), "\n",
                   "\t", "button_text_style:         ",
this.settings.button_text_style.toString(), "\n",
                   "\t", "button_text_top_padding:   ",
this.settings.button_text_top_padding.toString(), "\n",
                   "\t", "button_text_left_padding: ",
this.settings.button_text_left_padding.toString(), "\n",
                   "\t", "button_action:             ",
this.settings.button_action.toString(), "\n",
                   "\t", "button_disabled:           ",
this.settings.button_disabled.toString(), "\n",

                   "\t", "custom_settings:           ",
this.settings.custom_settings.toString(), "\n",
                   "Event Handlers:\n",
                   "\t", "swfupload_loaded_handler
assigned:  ", (typeof
this.settings.swfupload_loaded_handler ===
"function").toString(), "\n",
                   "\t", "file_dialog_start_handler
assigned: ", (typeof
this.settings.file_dialog_start_handler ===
"function").toString(), "\n",
                   "\t", "file_queued_handler assigned:
", (typeof this.settings.file_queued_handler ===
"function").toString(), "\n",
                   "\t", "file_queue_error_handler
assigned:  ", (typeof
this.settings.file_queue_error_handler ===
"function").toString(), "\n",
                   "\t", "upload_start_handler assigned:
", (typeof this.settings.upload_start_handler ===
"function").toString(), "\n",
                   "\t", "upload_progress_handler assigned:
", (typeof this.settings.upload_progress_handler ===
"function").toString(), "\n",
                   "\t", "upload_error_handler assigned:
", (typeof this.settings.upload_error_handler ===
"function").toString(), "\n",
                   "\t", "upload_success_handler assigned:
", (typeof this.settings.upload_success_handler ===
"function").toString(), "\n",
                    "\t", "upload_complete_handler assigned: 
", (typeof this.settings.upload_complete_handler === 
"function").toString(), "\n",
                    "\t", "debug_handler assigned: 
", (typeof this.settings.debug_handler === 
"function").toString(), "\n"
              ].join("")
       );
};

/* Note: addSetting and getSetting are no longer used by 
SWFUpload but are included
       the maintain v2 API compatibility
*/
// Public: (Deprecated) addSetting adds a setting value. If 
the value given is undefined or null then the default_value 
is used.
SWFUpload.prototype.addSetting = function (name, value, 
default_value) {
    if (value == undefined) {
        return (this.settings[name] = default_value);
    } else {
        return (this.settings[name] = value);
       }
};

// Public: (Deprecated) getSetting gets a setting. Returns 
an empty string if the setting was not found.
SWFUpload.prototype.getSetting = function (name) {
    if (this.settings[name] != undefined) {
        return this.settings[name];
       }

    return "";
};



// Private: callFlash handles function calls made to the 
Flash element.
// Calls are made with a setTimeout for some functions to 
work around
// bugs in the ExternalInterface library.
SWFUpload.prototype.callFlash = function (functionName, 
argumentArray) {
       argumentArray = argumentArray || [];
```

```
	var movieElement = this.getMovieElement();
	var returnValue, returnString;

	// Flash's method if calling ExternalInterface
methods (code adapted from MooTools).
	try {
		returnString =
movieElement.CallFunction('<invoke name="' + functionName +
'" returntype="javascript">' +
__flash__argumentsToXML(argumentArray, 0) + '</invoke>');
		returnValue = eval(returnString);
	} catch (ex) {
		throw "Call to " + functionName + " failed";
	}

	// Unescape file post param values
	if (returnValue != undefined && typeof
returnValue.post === "object") {
		returnValue =
this.unescapeFilePostParams(returnValue);
	}

	return returnValue;
};


/* *****************************
	-- Flash control methods --
	Your UI should use these
	to operate SWFUpload
   ***************************** */

// WARNING: this function does not work in Flash Player 10
// Public: selectFile causes a File Selection Dialog window
to appear.  This
// dialog only allows 1 file to be selected.
SWFUpload.prototype.selectFile = function () {
	this.callFlash("SelectFile");
};

// WARNING: this function does not work in Flash Player 10
// Public: selectFiles causes a File Selection Dialog
window to appear/ This
// dialog allows the user to select any number of files
// Flash Bug Warning: Flash limits the number of selectable
files based on the combined length of the file names.
```

```
// If the selection name length is too long the dialog will
fail in an unpredictable manner.  There is no work-around
// for this bug.
SWFUpload.prototype.selectFiles = function () {
	this.callFlash("SelectFiles");
};


// Public: startUpload starts uploading the first file in
the queue unless
// the optional parameter 'fileID' specifies the ID
SWFUpload.prototype.startUpload = function (fileID) {
	this.callFlash("StartUpload", [fileID]);
};

// Public: cancelUpload cancels any queued file.  The
fileID parameter may be the file ID or index.
// If you do not specify a fileID the current uploading
file or first file in the queue is cancelled.
// If you do not want the uploadError event to trigger you
can specify false for the triggerErrorEvent parameter.
SWFUpload.prototype.cancelUpload = function (fileID,
triggerErrorEvent) {
	if (triggerErrorEvent !== false) {
		triggerErrorEvent = true;
	}
	this.callFlash("CancelUpload", [fileID,
triggerErrorEvent]);
};

// Public: stopUpload stops the current upload and requeues
the file at the beginning of the queue.
// If nothing is currently uploading then nothing happens.
SWFUpload.prototype.stopUpload = function () {
	this.callFlash("StopUpload");
};

/* ************************
 * Settings methods
 *   These methods change the SWFUpload settings.
 *   SWFUpload settings should not be changed directly on
the settings object
 *   since many of the settings need to be passed to Flash
in order to take
 *   effect.
 * ********************** */
```

```
// Public: getStats gets the file statistics object.
SWFUpload.prototype.getStats = function () {
	return this.callFlash("GetStats");
};

// Public: setStats changes the SWFUpload statistics.  You
shouldn't need to
// change the statistics but you can.  Changing the
statistics does not
// affect SWFUpload accept for the successful_uploads count
which is used
// by the upload_limit setting to determine how many files
the user may upload.
SWFUpload.prototype.setStats = function (statsObject) {
	this.callFlash("SetStats", [statsObject]);
};

// Public: getFile retrieves a File object by ID or Index.
If the file is
// not found then 'null' is returned.
SWFUpload.prototype.getFile = function (fileID) {
	if (typeof(fileID) === "number") {
		return this.callFlash("GetFileByIndex",
[fileID]);
	} else {
		return this.callFlash("GetFile", [fileID]);
	}
};

// Public: addFileParam sets a name/value pair that will be
posted with the
// file specified by the Files ID.  If the name already
exists then the
// exiting value will be overwritten.
SWFUpload.prototype.addFileParam = function (fileID, name,
value) {
	return this.callFlash("AddFileParam", [fileID, name,
value]);
};

// Public: removeFileParam removes a previously set (by
addFileParam) name/value
// pair from the specified file.
SWFUpload.prototype.removeFileParam = function (fileID,
name) {
	this.callFlash("RemoveFileParam", [fileID, name]);
};
```

```
// Public: setUploadUrl changes the upload_url setting.
SWFUpload.prototype.setUploadURL = function (url) {
      this.settings.upload_url = url.toString();
      this.callFlash("SetUploadURL", [url]);
};

// Public: setPostParams changes the post_params setting
SWFUpload.prototype.setPostParams = function (paramsObject)
{
      this.settings.post_params = paramsObject;
      this.callFlash("SetPostParams", [paramsObject]);
};

// Public: addPostParam adds post name/value pair.  Each
name can have only one value.
SWFUpload.prototype.addPostParam = function (name, value) {
      this.settings.post_params[name] = value;
      this.callFlash("SetPostParams",
[this.settings.post_params]);
};

// Public: removePostParam deletes post name/value pair.
SWFUpload.prototype.removePostParam = function (name) {
      delete this.settings.post_params[name];
      this.callFlash("SetPostParams",
[this.settings.post_params]);
};

// Public: setFileTypes changes the file_types setting and
the file_types_description setting
SWFUpload.prototype.setFileTypes = function (types,
description) {
      this.settings.file_types = types;
      this.settings.file_types_description = description;
      this.callFlash("SetFileTypes", [types, description]);
};

// Public: setFileSizeLimit changes the file_size_limit
setting
SWFUpload.prototype.setFileSizeLimit = function
(fileSizeLimit) {
      this.settings.file_size_limit = fileSizeLimit;
      this.callFlash("SetFileSizeLimit", [fileSizeLimit]);
};
```

```
// Public: setFileUploadLimit changes the file_upload_limit
setting
SWFUpload.prototype.setFileUploadLimit = function
(fileUploadLimit) {
	this.settings.file_upload_limit = fileUploadLimit;
	this.callFlash("SetFileUploadLimit",
[fileUploadLimit]);
};

// Public: setFileQueueLimit changes the file_queue_limit
setting
SWFUpload.prototype.setFileQueueLimit = function
(fileQueueLimit) {
	this.settings.file_queue_limit = fileQueueLimit;
	this.callFlash("SetFileQueueLimit",
[fileQueueLimit]);
};

// Public: setFilePostName changes the file_post_name
setting
SWFUpload.prototype.setFilePostName = function
(filePostName) {
	this.settings.file_post_name = filePostName;
	this.callFlash("SetFilePostName", [filePostName]);
};

// Public: setUseQueryString changes the use_query_string
setting
SWFUpload.prototype.setUseQueryString = function
(useQueryString) {
	this.settings.use_query_string = useQueryString;
	this.callFlash("SetUseQueryString",
[useQueryString]);
};

// Public: setRequeueOnError changes the requeue_on_error
setting
SWFUpload.prototype.setRequeueOnError = function
(requeueOnError) {
	this.settings.requeue_on_error = requeueOnError;
	this.callFlash("SetRequeueOnError",
[requeueOnError]);
};

// Public: setHTTPSuccess changes the http_success setting
SWFUpload.prototype.setHTTPSuccess = function
(http_status_codes) {
	if (typeof http_status_codes === "string") {
		http_status_codes = http_status_codes.replace("
", "").split(",");
	}

	this.settings.http_success = http_status_codes;
	this.callFlash("SetHTTPSuccess",
[http_status_codes]);
};


// Public: setDebugEnabled changes the debug_enabled
setting
SWFUpload.prototype.setDebugEnabled = function
(debugEnabled) {
	this.settings.debug_enabled = debugEnabled;
	this.callFlash("SetDebugEnabled", [debugEnabled]);
};

// Public: setButtonImageURL loads a button image sprite
SWFUpload.prototype.setButtonImageURL = function
(buttonImageURL) {
	if (buttonImageURL == undefined) {
		buttonImageURL = "";
	}

	this.settings.button_image_url = buttonImageURL;
	this.callFlash("SetButtonImageURL",
[buttonImageURL]);
};

// Public: setButtonDimensions resizes the Flash Movie and
button
SWFUpload.prototype.setButtonDimensions = function (width,
height) {
	this.settings.button_width = width;
	this.settings.button_height = height;

	var movie = this.getMovieElement();
	if (movie != undefined) {
		movie.style.width = width + "px";
		movie.style.height = height + "px";
	}

	this.callFlash("SetButtonDimensions", [width,
height]);
};
```

```
// Public: setButtonText Changes the text overlaid on the
button
SWFUpload.prototype.setButtonText = function (html) {
      this.settings.button_text = html;
      this.callFlash("SetButtonText", [html]);
};
// Public: setButtonTextPadding changes the top and left
padding of the text overlay
SWFUpload.prototype.setButtonTextPadding = function (left,
top) {
      this.settings.button_text_top_padding = top;
      this.settings.button_text_left_padding = left;
      this.callFlash("SetButtonTextPadding", [left, top]);
};

// Public: setButtonTextStyle changes the CSS used to style
the HTML/Text overlaid on the button
SWFUpload.prototype.setButtonTextStyle = function (css) {
      this.settings.button_text_style = css;
      this.callFlash("SetButtonTextStyle", [css]);
};
// Public: setButtonDisabled disables/enables the button
SWFUpload.prototype.setButtonDisabled = function
(isDisabled) {
      this.settings.button_disabled = isDisabled;
      this.callFlash("SetButtonDisabled", [isDisabled]);
};
// Public: setButtonAction sets the action that occurs when
the button is clicked
SWFUpload.prototype.setButtonAction = function
(buttonAction) {
      this.settings.button_action = buttonAction;
      this.callFlash("SetButtonAction", [buttonAction]);
};

// Public: setButtonCursor changes the mouse cursor
displayed when hovering over the button
SWFUpload.prototype.setButtonCursor = function (cursor) {
      this.settings.button_cursor = cursor;
      this.callFlash("SetButtonCursor", [cursor]);
};

/* *******************************
      Flash Event Interfaces
      These functions are used by Flash to trigger the
various
      events.
```

```
	All these functions a Private.

	Because the ExternalInterface library is buggy the
event calls
	are added to a queue and the queue then executed by a
setTimeout.
	This ensures that events are executed in a
determinate order and that
	the ExternalInterface bugs are avoided.
******************************* */

SWFUpload.prototype.queueEvent = function (handlerName,
argumentArray) {
	// Warning: Don't call this.debug inside here or
you'll create an infinite loop

	if (argumentArray == undefined) {
		argumentArray = [];
	} else if (!(argumentArray instanceof Array)) {
		argumentArray = [argumentArray];
	}

	var self = this;
	if (typeof this.settings[handlerName] === "function")
{
		// Queue the event
		this.eventQueue.push(function () {
			this.settings[handlerName].apply(this,
argumentArray);
		});

		// Execute the next queued event
		setTimeout(function () {
			self.executeNextEvent();
		}, 0);

	} else if (this.settings[handlerName] !== null) {
		throw "Event handler " + handlerName + " is
unknown or is not a function";
	}
};

// Private: Causes the next event in the queue to be
executed.  Since events are queued using a setTimeout
// we must queue them in order to garentee that they are
executed in order.
```

```
SWFUpload.prototype.executeNextEvent = function () {
	// Warning: Don't call this.debug inside here or
you'll create an infinite loop

	var  f = this.eventQueue ? this.eventQueue.shift() :
null;
	if (typeof(f) === "function") {
		f.apply(this);
	}
};

// Private: unescapeFileParams is part of a workaround for
a flash bug where objects passed through ExternalInterface
cannot have
// properties that contain characters that are not valid
for JavaScript identifiers. To work around this
// the Flash Component escapes the parameter names and we
must unescape again before passing them along.
SWFUpload.prototype.unescapeFilePostParams = function
(file) {
	var reg = /[$]([0-9a-f]{4})/i;
	var unescapedPost = {};
	var uk;

	if (file != undefined) {
		for (var k in file.post) {
			if (file.post.hasOwnProperty(k)) {
				uk = k;
				var match;
				while ((match = reg.exec(uk)) !==
null) {
					uk = uk.replace(match[0],
String.fromCharCode(parseInt("0x" + match[1], 16)));
				}
				unescapedPost[uk] = file.post[k];
			}
		}

		file.post = unescapedPost;
	}

	return file;
};

// Private: Called by Flash to see if JS can call in to
Flash (test if External Interface is working)
SWFUpload.prototype.testExternalInterface = function () {
	try {
		return this.callFlash("TestExternalInterface");
	} catch (ex) {
		return false;
	}
};

// Private: This event is called by Flash when it has
finished loading. Don't modify this.
// Use the swfupload_loaded_handler event setting to
execute custom code when SWFUpload has loaded.
SWFUpload.prototype.flashReady = function () {
	// Check that the movie element is loaded correctly
with its ExternalInterface methods defined
	var movieElement = this.getMovieElement();

	if (!movieElement) {
		this.debug("Flash called back ready but the
flash movie can't be found.");
		return;
	}

	this.cleanUp(movieElement);

	this.queueEvent("swfupload_loaded_handler");
};

// Private: removes Flash added fuctions to the DOM node to
prevent memory leaks in IE.
// This function is called by Flash each time the
ExternalInterface functions are created.
SWFUpload.prototype.cleanUp = function (movieElement) {
	// Pro-actively unhook all the Flash functions
	try {
		if (this.movieElement &&
typeof(movieElement.CallFunction) === "unknown") { // We
only want to do this in IE
			this.debug("Removing Flash functions
hooks (this should only run in IE and should prevent memory
leaks)");
			for (var key in movieElement) {
				try {
					if
(typeof(movieElement[key]) === "function") {
						movieElement[key] =
null;
					}
					} catch (ex) {
					}
				}
			}
		} catch (ex1) {

		}

		// Fix Flashes own cleanup code so if the SWFMovie
was removed from the page
		// it doesn't display errors.
		window["__flash__removeCallback"] = function
(instance, name) {
			try {
				if (instance) {
					instance[name] = null;
				}
			} catch (flashEx) {

			}
		};

};


/* This is a chance to do something before the browse
window opens */
SWFUpload.prototype.fileDialogStart = function () {
	this.queueEvent("file_dialog_start_handler");
};


/* Called when a file is successfully added to the queue.
*/
SWFUpload.prototype.fileQueued = function (file) {
	file = this.unescapeFilePostParams(file);
	this.queueEvent("file_queued_handler", file);
};


/* Handle errors that occur when an attempt to queue a file
fails. */
SWFUpload.prototype.fileQueueError = function (file,
errorCode, message) {
	file = this.unescapeFilePostParams(file);
	this.queueEvent("file_queue_error_handler", [file,
errorCode, message]);
```

```
};

/* Called after the file dialog has closed and the selected
files have been queued.
	You could call startUpload here if you want the
queued files to begin uploading immediately. */
SWFUpload.prototype.fileDialogComplete = function
(numFilesSelected, numFilesQueued) {
	this.queueEvent("file_dialog_complete_handler",
[numFilesSelected, numFilesQueued]);
};

SWFUpload.prototype.uploadStart = function (file) {
	file = this.unescapeFilePostParams(file);
	this.queueEvent("return_upload_start_handler", file);
};

SWFUpload.prototype.returnUploadStart = function (file) {
	var returnValue;
	if (typeof this.settings.upload_start_handler ===
"function") {
		file = this.unescapeFilePostParams(file);
		returnValue =
this.settings.upload_start_handler.call(this, file);
	} else if (this.settings.upload_start_handler !=
undefined) {
		throw "upload_start_handler must be a
function";
	}

	// Convert undefined to true so if nothing is
returned from the upload_start_handler it is
	// interpretted as 'true'.
	if (returnValue === undefined) {
		returnValue = true;
	}

	returnValue = !!returnValue;

	this.callFlash("ReturnUploadStart", [returnValue]);
};



SWFUpload.prototype.uploadProgress = function (file,
bytesComplete, bytesTotal) {
	file = this.unescapeFilePostParams(file);
	this.queueEvent("upload_progress_handler", [file,
bytesComplete, bytesTotal]);
};

SWFUpload.prototype.uploadError = function (file,
errorCode, message) {
	file = this.unescapeFilePostParams(file);
	this.queueEvent("upload_error_handler", [file,
errorCode, message]);
};

SWFUpload.prototype.uploadSuccess = function (file,
serverData) {
	file = this.unescapeFilePostParams(file);
	this.queueEvent("upload_success_handler", [file,
serverData]);
};

SWFUpload.prototype.uploadComplete = function (file) {
	file = this.unescapeFilePostParams(file);
	this.queueEvent("upload_complete_handler", file);
};

/* Called by SWFUpload JavaScript and Flash functions when
debug is enabled. By default it writes messages to the
   internal debug console.  You can override this event and
have messages written where you want. */
SWFUpload.prototype.debug = function (message) {
	this.queueEvent("debug_handler", message);
};


/* **********************************
	Debug Console
	The debug console is a self contained, in page
location
	for debug message to be sent.  The Debug Console adds
	itself to the body if necessary.

	The console is automatically scrolled as messages
appear.

	If you are using your own debug handler or when you
deploy to production and
	have debug disabled you can remove these functions to
reduce the file size
	and complexity.
```

```
********************************** */

// Private: debugMessage is the default debug_handler.  If
you want to print debug messages
// call the debug() function.  When overriding the function
your own function should
// check to see if the debug setting is true before
outputting debug information.
SWFUpload.prototype.debugMessage = function (message) {
	if (this.settings.debug) {
		var exceptionMessage, exceptionValues = [];

		// Check for an exception object and print it
nicely
		if (typeof message === "object" && typeof
message.name === "string" && typeof message.message ===
"string") {
			for (var key in message) {
				if (message.hasOwnProperty(key)) {
					exceptionValues.push(key +
": " + message[key]);
				}
			}
			exceptionMessage =
exceptionValues.join("\n") || "";
			exceptionValues =
exceptionMessage.split("\n");
			exceptionMessage = "EXCEPTION: " +
exceptionValues.join("\nEXCEPTION: ");

	SWFUpload.Console.writeLine(exceptionMessage);
		} else {
			SWFUpload.Console.writeLine(message);
		}
	}
};

SWFUpload.Console = {};
SWFUpload.Console.writeLine = function (message) {
	var console, documentForm;

	try {
		console =
document.getElementById("SWFUpload_Console");

		if (!console) {
```

```
                    documentForm =
document.createElement("form");

      document.getElementsByTagName("body")[0].appendChild(
documentForm);

                    console =
document.createElement("textarea");
                    console.id = "SWFUpload_Console";
                    console.style.fontFamily = "monospace";
                    console.setAttribute("wrap", "off");
                    console.wrap = "off";
                    console.style.overflow = "auto";
                    console.style.width = "700px";
                    console.style.height = "350px";
                    console.style.margin = "5px";
                    documentForm.appendChild(console);
             }

             console.value += message + "\n";

             console.scrollTop = console.scrollHeight -
console.clientHeight;
      } catch (ex) {
             alert("Exception: " + ex.name + " Message: " +
ex.message);
      }
};
```